SSC18-X-01

A Dynamic Open Source Long Term Archiving and Trending Solution for Small Satellites

Ryan Melton, Jason Thomas
Ball Aerospace
1600 Commerce St, Boulder, CO 80301; 303-939-6771
rmelton@ball.com

ABSTRACT

Archiving and trending data is a universal problem for satellite operators. Traditionally this functionality is coded from scratch when creating a ground software solution for a satellite program. This is time consuming, costly and impractical for the typically cost constrained small satellite environment. The open source COSMOS C2 system has eased the burden on small satellite operators for over three years. Its emphasis on simple text file based configuration and ease of use has enabled dramatic cost and time savings across the industry. However, until recently it did not have an included archiving and trending tool. To analyze telemetry, users had to open individual telemetry log files which made trending over time periods of days, weeks, and months very time consuming if not altogether impractical. That has changed with the introduction of DART (Data Archival Retrieval and Trending). DART utilizes a SQL database and maintains the original COSMOS telemetry files while providing instant access to decommutated data over long time periods. DART synchronizes with the COSMOS server and stays up to date even as telemetry definitions change and data points are added and removed. This paper discusses the need and use cases for a data archiving and trending solution in the small satellite environment. It outlines the architectural and implementation details which went into the creation of the DART tool. Finally, it explores the tool in action with practical use cases using real telemetry data.

INTRODUCTION

Every satellite generates telemetry data indicating status such as internal temperatures, voltage levels, command counters, events, and various mission specific values. This data is not difficult to manage over small time periods, especially when using a command and control system that can easily parse data logged into files. However, over periods of weeks, months, and years, megabytes become gigabytes, and for many modern systems, eventually terabytes of data. Running queries that require reading through and parsing vast troves of data at the gigabyte and terabyte level quickly becomes impractical when fast answers to questions are desired.

COSMOS DART (Data Archiving and Trending) was created to quickly retrieve data that was captured over a large period, and to remove the burden of projects having to track, catalog, and organize hundreds of files of logged data. With DART, all data is ingested into a single system and is indexed by time. Additionally, numerical data is automatically reduced into a mean, maximum, minimum, and standard deviation for time intervals of minutes, hours, and days. This reduced data set allows for instantaneous answers to questions such as “What was the maximum temperature of the sensor over the last year?”.

DART also logs all commands which allows the correlation of commanding to changes in telemetry.

DART is a free and fully open source solution as part of Ball Aerospace COSMOS versions 4.2+ which was released in April 2018.

OPTIMAL USE CASES

DART solves the most common problems encountered when needing to parse through large amounts of logged data.

Historical Questions

The simplest use case for DART is to handle arbitrary questions such as “How many image captures did the satellite take last week?”. To answer this question, a user would give DART the time at the beginning of the desired week, the time at the end of the desired week, and the name of the image collect counter telemetry item. DART would then return an array of values of the counter over the week and the user could subtract the last and first value to determine the number of collects.

Other related questions would be: “What was the smallest value of a telemetry point?”, “What was the largest?”, “What was the average value over the last week?”

Anomaly Investigation

One of the most important use cases for DART is being able to quickly investigate an anomaly. When something goes wrong on orbit, DART can be used to gather the values of relevant telemetry points both leading up to and after the occurrence of an anomaly.

Anomaly Prediction

The holy grail of data archiving systems is to be able to predict future anomalies before they occur. DART does not do this automatically, but it can be very useful for noticing historical trends that may be predictive of future failures. For example, by periodically plotting the current draw from a motor and noticing an upward trend, you might be able to predict growing friction within the motor and a future failure.

Limited Life Tracking

Some programs have requirements to accurately monitor the run time of satellite components. By querying the power on / off status of a component over time, DART can be used to calculate the total run time for the component.

Scenario Playback

DART stores every packet that is received during a contact or test activity. Through its integration with COSMOS Replay, DART can be used to playback logged packets as if they were being received in realtime. This can be very useful for operator training and other scenarios where it's beneficial to have realtime telemetry screens.

UNOPTIMIZED USE CASES

DART is optimized to retrieve the values of one or more items over a large period. However, it is not optimized for all use cases. Particularly, DART is not optimized for comparing one item against another. This can still be done by querying both items and then performing your own comparisons on the retrieved data in code or in a tool such as Excel. However, DART does not provide this kind of querying natively.

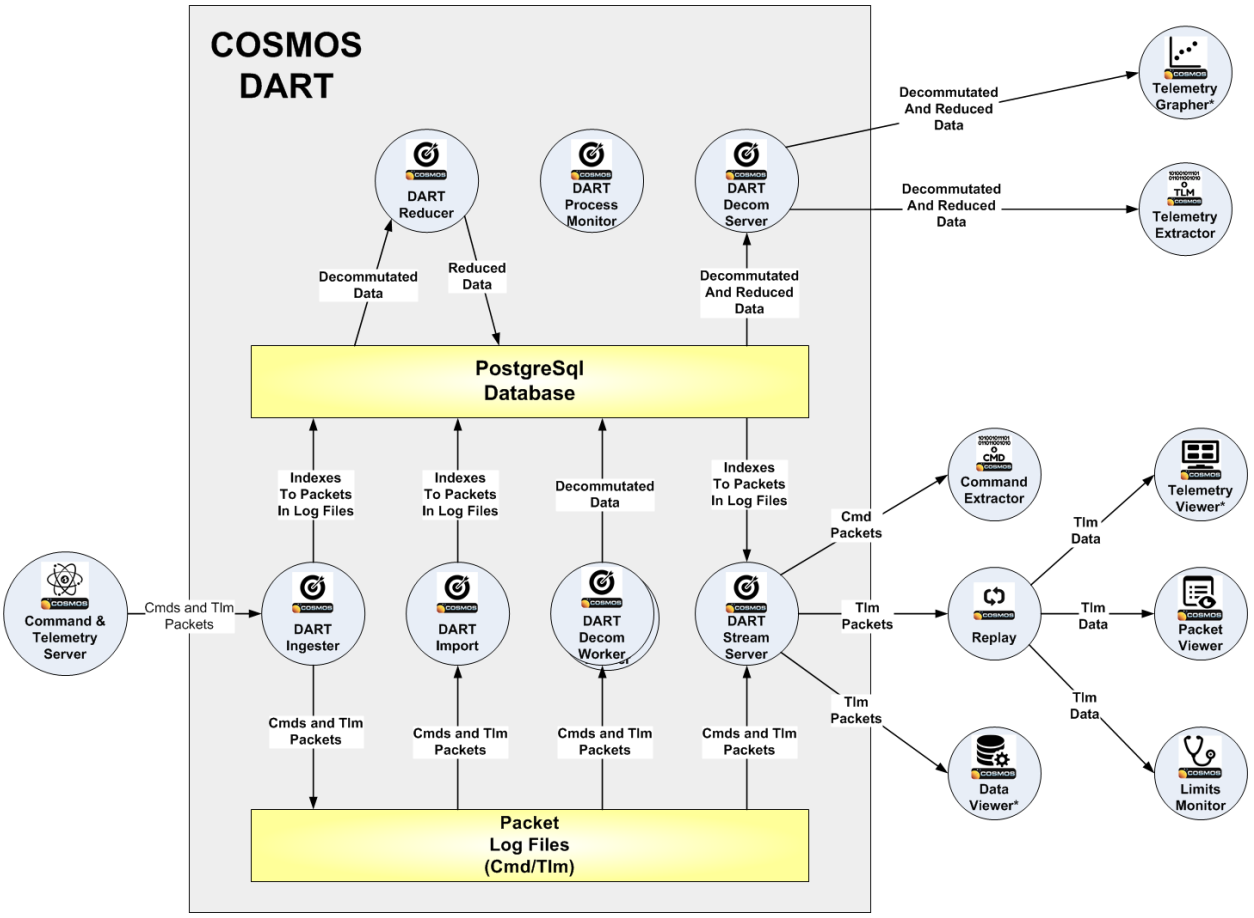

Figure 1: DART Architecture

ARCHITECTURE

DART is composed of seven internal applications depicted as circles within the colored square in Figure 1 above. Whenever DART is running all internal applications are also running except for DART Import which is only used to manually ingest log files in off-nominal scenarios. The following paragraphs describe each of these applications.

DART Process Monitor

This is the "main" DART process that starts all the other processes that make up the complete DART system. When the DART Process Monitor first starts up it performs and optimized algorithm to clean the database from any inconsistencies that may have occurred from a previous harsh shutdown. It then starts the next five applications below and monitors their aliveness. If any monitored process unexpectedly dies, then the DART process monitor automatically restarts it. Shutting down the process monitor also properly shuts down the other DART applications.

DART Ingester

The DART Ingester is responsible for logging raw packet data into DART as it is received by the main COSMOS System. At startup, it connects to the COSMOS CmdTlmServer preidentified command and telemetry routers and receives all realtime command and telemetry packets across a TCP/IP stream. As each packet is received, it is logged into a binary log file, and a PacketLogEntry database item is created that indexes where in the file that packet is stored.

Any data that is captured by the COSMOS CmdTlmServer when the DART Ingester is not running can be added into the database later using the DART Importer tool as discussed later.

DART Decom Workers

The DART Decom Worker processes decommutate raw packets by breaking them down into their individual items and values. It then saves the decommutated data into the PostgreSQL database. An environment variable can be used to control how many Decom Workers are spawned, which currently defaults to 2. The DART Decom Workers are the most processor intensive part of DART and the number of workers may need to be scaled based on bit rates and the complexity of the system streaming realtime data to DART.

DART Reducer

The DART Reducer process reduces decommutated data into minute/hour/day reduced data sets. Each item that can be reduced is broken down into a maximum, minimum, average, and standard deviation over the representative period. Only numerical items are reduced by DART. Due to ordering constraints, the DART Reducer is implemented as a single process with a variable number of threads.

DART Stream Server

The DART Stream Server provides a stream of raw packets from DART on request. Requests typically specify a time range, and optionally can limit the stream to a specific set of packets. The full API for the DART Stream server is discussed in a later section.

The DART Stream Server is used for tools that need complete raw packets of data and every bit in each packet available to them.

DART Decom Server

The DART Decom server provides a JSON formatted array of decommutated or reduced data for an item on request. Again, requests typically specify a time range. Notably the DART Decom Server only returns data for one item at a time. Data for multiple items can be retrieved using multiple queries. The full API for the DART Decom server is discussed in a later section.

The DART Decom Server is used by tools that need quick lookups of specific item values over time.

DART Import

DART Import is a command line tool to import previously logged data into DART. This is useful for importing older data that was never imported such as if DART was offline when the data was collected. Important: These log files are imported in place and become part of the DART database. The files must be placed into the DART data folder (defaults to outputs/dart/data) before they can be imported. Note that the data requires a SYSTEM META packet, and therefore generally only supports data from COSMOS 4.1.1+ unless the data is massaged first.

Note that in nominal use, the COSMOS Command and Telemetry Server streams data to DART in realtime and DART Import is not required.

USER INTERFACE & COSMOS INTEGRATION

The primary User Interface for DART are the existing COSMOS tools that can handle data logged into files. Additional user interfaces can easily be created using the provided DART APIs.

For decommutated and reduced data, COSMOS TlmGrapher and TlmExtractor can pull data from the DART Decom Server. TlmGrapher provides a line graph style display of data, and TlmExtractor can provide that same data as a CSV file.

For streaming raw packets, Replay, DataViewer, and CmdExtractor can pull data from the DART Stream Server. DataViewer provides a scrolling text display of data which is particularly useful for viewing memory dumps and event style data. CmdExtractor pulls the details of when commands are sent and the parameters involved in each command. COSMOS Replay allows the other COSMOS tools that only support realtime telemetry to receive data from DART (as if it was in realtime).

DATABASE

PostgreSQL was chosen as the database engine for DART. COSMOS is a fully open source system so choosing an open source database that users could easily install and experiment with on their desktop machines was an important requirement. MySQL and PostgreSQL are the primary contenders for open source SQL databases. PostgreSQL's native support for storing array data and JSON data lead to its choice. Time series databases and object store databases were also considered, but none were found that were of sufficient ease of use, maturity, or built in indexing capabilities.

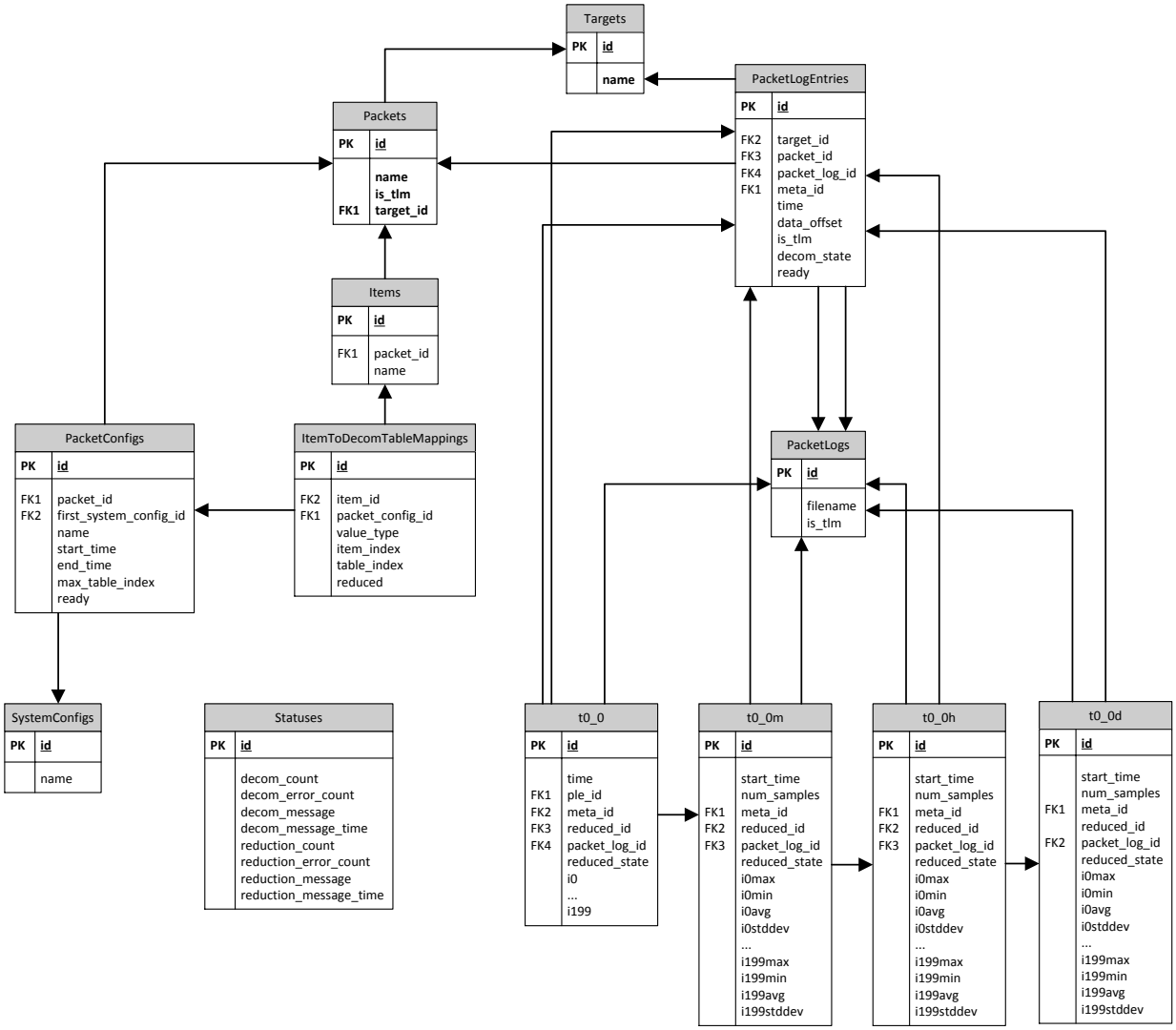

Figure 2: DART Schema

SCHEMA DESIGN

The DART schema was designed with two primary requirements: fast queries, and adaptability. DART is designed to be able to provide quick answers to queries even over years of stored data. This is accomplished by a simple time organized schema and automatic data reduction.

Adaptability is equally important. Especially during development, the contents and number of command and telemetry packets in a system evolve. Fields are added and others are deleted. A fully functional data archive must be able to adapt to these changes.

The following tables exist in the DART database. Each table has an implied primary key called "id" which is a 64-bit integer. The data type Integer is used loosely below for many different sizes of Integer. Primary keys are always big integers (64-bit) and foreign keys that reference them are big integers for tables that are expected to have many entries (such as PacketLogEntries, and data entries).

Targets

The Targets table contains the name of all the COSMOS targets that have data stored in the DART database. This table only changes when a new target is discovered by DART.

Field Name	Data Type	Notes

Name	String	Holds the name of the COSMOS Target

Packets

The Packets table contains the name of each command and telemetry packet defined in DART. This table only changes when a new command or telemetry packet is discovered by DART.

Field Name	Data Type	Notes
Target_id	Integer	Foreign key to the packet's target in the Targets table
Name	String	Holds the name of the COSMOS Packet
Is_tlm	Boolean	Flag that is true for telemetry packets and false for command packets

Items

The Items table contains the name of each item in each packet. This table only changes when a new item is discovered in a command or telemetry packet.

Field Name	Data Type	Notes
Packet_id	Integer	Foreign key to the item's packet in the Packets table
Name	String	Holds the name of the item in the packet

Packet Logs

The Packet Logs table keeps track of the files containing raw packet data managed by DART. A new entry is created each time a new log file is created. New files are created by default every 2GB of data logged and every time DART restarts.

Field Name	Data Type	Notes
Filename	Integer	Full path to a raw packet file managed by DART
Is_tlm	Boolean	Flag that is true for a log file containing telemetry packets and false for a log file of command packets

Packet Log Entries

The Packet Log Entries table provides an entry for every packet in a packet log file. Raw packets are stored in the normal COSMOS packet log files, because SQL databases are not well suited for storing large amounts of binary blob type data. Packet Log Entries contain the offset into the file to find each packet and are used as an index for random access to raw packet data.

Field Name	Data Type	Notes
Target_id	Integer	Foreign key to the packet's target name in the Targets table
Packet_id	Integer	Foreign key to the packet's packet name in the Packet table
Packet_log_id	Integer	Foreign key to the packet log that the packet is stored in
Meta_id	Integer	Foreign key to the COSMOS SYSTEM META packet relevant to this packet log entry
Time	Datetime	Timestamp of the packet log entry
Data_offset	Integer	Offset into the packet log file to the beginning of this packet
Is_tlm	Boolean	Flag indicating if this is a command or telemetry packet. True = telemetry, false = command
Decom_state	Integer	State indicating if this packet has yet been decommutated by a DART worker application
Ready	Boolean	Flag when true indicating that this packet has been flushed to disk and is ready to be read from the packet log file.

System Configs

The System configs table contains the name of each system config DART is aware of. COSMOS System configs are a full COSMOS configuration at a given point of time (all the command and telemetry packet definitions). Each system config is named with a 32-byte md5sum that is calculated over the corresponding COSMOS configuration files.

Field	Data Type	Notes

Name		
Name	String	Holds the name of the COSMOS System Configuration

Packet Configs

The packet configs table tracks a specific configuration for each packet. Within each system configuration, is the definition of every packet, but between system configurations a packet may not change in any way. The packet configuration names are a unique identifier for the specific definition of a packet. If the contents of a packet do not change between system configurations, DART does not need to create new tables to hold the packet's data. Each entry in this table tracks the creation of the specific table needed to store the data for a given packet configuration as well as the time range of data that DART is storing for this packet configuration.

Field Name	Data Type	Notes
Packet_id	Integer	Foreign key to the packet config's packet in the Packets table
First_system_config_id	Integer	Foreign key to the first system configuration that contained this packet config
Name	String	Holds the name of the packet configuration (32byte md5sum)
Start_time	Datetime	Time of the earliest packet in DART with this packet configuration
End_time	Datetime	Time of the most recent packet in DART with this packet configuration
Max_table_index	Integer	Highest index of the subtables that store data for this packet configuration
Ready	Boolean	Flag indicating that all the subtables that store data for this packet configuration have been successfully created

Item to Decom Table Mappings

The item to decom table mappings table contains the information needed to find the data for a specific item in a packet configuration. An entry into this table is created for every item in every packet for every packet config. Items with both a raw and converted representation will have two entries.

Field Name	Data Type	Notes
Item_id	Integer	Foreign key to the item that is represented in the data table
Packet_config_id	Integer	Foreign key to the packet configuration that goes with this data table
Value_type	Integer	Enumerated type indicating if this entry represents the RAW, or CONVERTED value of an item. May also be RAW_CON which means the raw and converted values are the same.
Item_index	Integer	Index of the item in the data table (0 based)
Table_index	Integer	Index of the data table (0 based)
Reduced	Boolean	Indicates if the value is further reduced after being decommutated

Data Tables

DART dynamically creates data tables whenever a new packet config is created. Decommutated data tables are named tX_Y in PostgreSQL where X is an integer that matches the packet configuration id and Y is an integer that counts up from 0 for each subtable. Data tables are limited to 200 items per subtable, so any packet with more than 200 items (or item types with RAW/CONVERTED) data will be broken up into 2 or more subtables. For small packets Y will always be 0. Reduced tables are named correspondingly as tX_Ym for minute reduced data, tX_Yh for hour reduced data, and tX_Yd for day reduced data.

Every item in a packet is given a new name of Ix where x is just a number that increments with each item in the packet or data type. The Item to Decom Mapping table

is used to map between real item names and the names in the data tables.

The decommutated tables take the following form:

Field Name	**Data Type**	**Notes**
Time	Datetime	The time of the packet that was decommutated
Ple_id	Integer	Foreign key to the packet log entry that this data was decommutated from
Meta_id	Integer	Foreign key to the COSMOS SYSTEM META packet log entry that relates to this entry
Reduced_id	Integer	Foreign key to the minute reduced data produced from this data
Packet_log_id	Integer	Foreign key to the packet log that this raw data is stored in
Reduced_state	Integer	State indicating if this data has yet been reduced by the DART Reducer application
I<First Index>	Varies	Either the raw or converted value of the first item in the packet.
…	…	…
I<Last Index>	Varies	Either the raw or converted value of the last item in the packet, or the last item that fit within this specific subtable.

The reduced tables differ slightly because they are derived from the decommutated data, and take the following form:

Field Name	**Data Type**	**Notes**
Start_time	Datetime	The time of the first item that was used to create this reduction
Num_samples	Integer	Number of samples that were used to create this reduction
Meta_id	Integer	Foreign key to the COSMOS SYSTEM META packet log entry that relates to this entry
Reduced_id	Integer	Foreign key to the hour or day reduced data produced from this data
Packet_log_id	Integer	Foreign key to the packet log that this raw data is stored in
Reduced_state	Integer	State indicating if this data has yet been reduced by the DART Reducer application
I<First Index>max	Varies	Either the raw or converted maximum value of the first item in the packet.
I<First Index>min	Varies	Either the raw or converted minimum value of the first item in the packet.
I<First Index>avg	Varies	Either the raw or converted average value of the first item in the packet.
I<First Index>stddev	Varies	Either the raw or converted standard deviation value of the first item in the packet.
…	…	…
I<Last Index>max	Varies	Either the raw or converted maximum value of the last item in the packet or last item in this subtable.
I<Last Index>min	Varies	Either the raw or converted minimum value of the last item in the packet or last item in this subtable..
I<Last Index>avg	Varies	Either the raw or converted average value of the last item in the packet or last item in this subtable..

I<Last Index>stddev	Varies	Either the raw or converted standard deviation value of the last item in the packet or last item in this subtable..

Statuses

The statuses table provides a location for the DART applications to provide status to the outside world (besides writing messages to log files). There is only one row in this table that contains the latest status values.

Field Name	**Data Type**	**Notes**
Decom_count	Integer	Count of packets decommutated
Decom_error_count	Integer	Count of errors decommutating packets
Decom_message	Text	Most recent message from a decommutation worker application
Decom_message_time	Datetime	Time of the decommutation message
Reduction_count	Integer	Count of packets reduced
Reduction_error_count	Integer	Count of errors reducing packets
Reduction_message	Text	Most recent message from the reduction process
Reduction_message_time	Datetime	Time of the reduction message

PERFORMANCE / SIZE TRADES

DART has been optimized to provide quick time based queries of command and telemetry data. Providing quick queries involves a tradeoff between storage overhead and speed of the query. Database indexes are the natural way to improve the performance of queries in modern SQL database systems such as PostgreSQL. Unfortunately indexes also take up significant disk space and can slow down writes to tables.

During development of DART it was discovered that indexes often reached up to one third of the size of the table they were indexing. Therefore, only indexes absolutely required to ensure fast performance for the normal types of DART queries were created (primarily time).

For specific cases such as reduced_state in the data tables, PostgreSQL partial indexes were utilized to only store the necessary indexes to rows that still needed to be reduced (and ignore rows that are already reduced).

API DESIGN

DART provides two different APIs, one to retrieve raw packet data as a stream, and another to request up to 10,000 data points at a time of decommutated or reduced data. These APIs are used directly by the COSMOS tools, but may also be used by any other custom software.

Raw Packet API

The raw packet API streams raw packets over a TCP/IP connection. DART acts as the TCP/IP server, by default listening on port 8779. Data is transferred in the stream using the COSMOS 4 preidentified stream protocol.

The COSMOS 4 preidentified stream protocol puts the following header before each packet that is transmitted.

Field Name	**Data Type**	**Notes**
Flags	1-byte	Bit 0 (left most significant) bit is a flag that indicates if the packet is stored telemetry (if set) or realtime telemetry (if not set). Bit 1 is flag indiciting if the extra header is present (if set) or not (if not set)
Extra_length	4-byte unsigned integer	(Optional based on flags) Length of following extra data
Extra_data	Extra_length-bytes of data	(Optional based on flags) JSON encoded hash of extra data associated with this packet.

Time_seconds	4-byte integer	COSMOS received time seconds of the packet (unix epoch)
Time_microseconds	4-byte integer	COSMOS received time microseconds of the packet (unix epoch)
Target_name_length	1-byte unsigned integer	Length of the packet's target name string
Target_name	Target_name_length string	The name of the packet's target in COSMOS
Packet_name_length	1-byte unsigned integer	Length of the packet's packet name string
Packet_name	Packet_name_length string	The name of the packet in COSMOS
Packet_data_length	4-byte unsigned integer	Length of the packet's data
Packet_data	Packet_data_length bytes	The data that makes up the raw binary packet

To request a stream of raw packets, the client needs to connect to the TCP/IP port and then send a message with the above format, using DART as the target name and DART as the packet name. The packet data should be formatted as a JSON string with the following fields:

Field Name	**Data Type**	**Notes**
start_time_sec	Integer	Unix start time in seconds (UTC) Example: 1514764800 (Jan 1, 2018 00:00:00)
start_time_usec	Integer	Microseconds part of the start time. Example: 0
end_time_sec	Integer	Unix end time in seconds (UTC) Example: 1514809815 (Jan 1, 2018 12:30:15)
end_time_usec	Integer	Microseconds part of the end time. Example: 0
packets	Array of Arrays	Array of arrays specifying the target and packet name Example: [['INST', 'HEALTH_STATUS'], ['INST', 'ADCS']] Optional. If not given defaults to all packets
cmd_tlm	String	Whether the request is for command or telemetry data Options: 'CMD', 'TLM', Default: 'TLM'
meta_filters	Array of Strings	Array of logical meta data filter expressions. Supports logical assertions on all the defined SYSTEM META items in your COSMOS definition. Logical operators include '=' (or '==', both mean equals), '!= (not equal)', '>', '<', '>=', and '<='. Example: ["OPERATOR_NAME" == "Jason"], Default: []
meta_ids	Array of Integers	Array of the meta ID(s) to use when filtering the data. The meta IDs are an internal DART ID and thus this is only used if you have obtained the database meta IDs from a previous DART Decommutation Server request. Example: [1], Default: []

Simple Example JSON Data:

{"start_time_sec": 1514764800, "start_time_usec": 0, "end_time_sec": 1514809815, "end_time_usec": 0}

The server will then respond with a stream of the requested packets and automatically close the socket when all packets have been sent.

Decommutated Data API

The decommutated data API follows a single command / response model and as such uses the same JSON-RPC over HTTP protocol as used by the main COSMOS CmdTlmServer API. See the COSMOS JSON-API documentation for more details. http://cosmosrb.com/docs/json_api/

The following fields are sent to the server over HTTP by default at port 8777. The method name should be 'query'.

Field Name	**Data Type**	**Notes**

start_time_sec	Integer	Unix start time in seconds (UTC) Example: 1514764800 (Jan 1, 2018 00:00:00)
start_time_usec	Integer	Microseconds part of the start time. Example: 0
end_time_sec	Integer	Unix end time in seconds (UTC) Example: 1514809815 (Jan 1, 2018 12:30:15)
end_time_usec	Integer	Microseconds part of the end time. Example: 0
item	Array with three strings	Array specifying the target, packet, and item name Example: ['INST', 'HEALTH_STATUS', 'TEMP1']
reduction	String	How to reduce the data. Options: "NONE", "MINUTE", "HOUR", "DAY"
value_type	String	The type of data to return. Options: "RAW", "RAW_MAX", "RAW_MIN", "RAW_AVG", "RAW_STDDEV", "CONVERTED", "CONVERTED_MAX", "CONVERTED_MIN", "CONVERTED_AVG", "CONVERTED_STDDEV"
cmd_tlm	String	Whether the request is for command or telemetry data Options: 'CMD', 'TLM', Default: 'TLM'
limit	Integer	Maximum number of data items to return. Must be less than or equal to 10000. Default: 10000
offset	Integer	Offset into the data stream. Since the maximum number of values allowed is 10000, you can set the offset to 10000, then 20000, etc to get additional values Example: 10000, Default: 0
meta_filters	Array of Strings	Array of logical meta data filter expressions. Supports logical assertions on all the defined SYSTEM META items in your COSMOS definition. Logical operators include '=' (or '==', both mean equals), '!= (not equal)', '>', '<', '>=', and '<='. Example: ["OPERATOR_NAME" == "Jason"], Default: []
meta_ids	Array of Integers	Array of the meta ID(s) to use when filtering the data. The meta IDs are an internal DART ID and thus this is only used if you have obtained the database meta IDs from a previous DART Decommutation Server request. Example: [1], Default: []

After sending the request, the JSON RPC response will contain an Array of Arrays containing the item value, item seconds, item microseconds, samples (always 1 for NONE reduction, varies for other reduction values), and meta_id. Note this meta_id is the ID which can be used in subsequent requests in the meta_ids field.

Simple Example Request and Response:

--> {"jsonrpc": "2.0", "method": "query", "params": [{"start_time_sec": 1514764800, "start_time_usec": 0, "end_time_sec": 415000000, "end_time_usec": 0, "item": ["INST", "HEALTH_STATUS", "TEMP1"], "reduction": "NONE", "value_type": "CONVERTED"}], "id": 1}

<-- {"jsonrpc": "2.0", "result": [[10.3, 1514764800, 0, 1, 1], [15.6, 1514764801, 340, 1, 1]], "id": 1}

SETUP

Setting up DART is very simple. If COSMOS is already configured for your project, then all that remains is to create a user and database in Postgres and then run two commands in a terminal.

Detailed installation instructions for installing and setting up DART can be found at: http://cosmosrb.com/docs/dart_install/

SUMMARY

DART provides a high performance open source solution for managing large amounts of command and telemetry data for small satellite programs.

For more information and to get started with Ball Aerospace COSMOS and DART please see http://cosmosrb.com.

REFERENCES

1. Melton, Ryan, "Ball Aerospace COSMOS" *Retrieved from http://cosmosrb.com June 1st, 2018*
2. The PostgreSQL Global Development Group, *"PostgreSQL" Retrieved from* https://www.postgresql.org *June 1st, 2018*
3. *JSON-RPC, "JSON-RPC" Retrieved from* http://www.jsonrpc.org/specification *June 1st, 2018*